VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

NÁSTROJ PRO KONTROLU DODRŽOVÁNÍ UX PRINCIPŮ

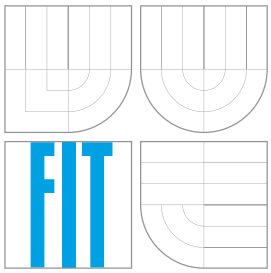
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

VÍTĚZSLAV KŘÍŽ

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

NÁSTROJ PRO KONTROLU DODRŽOVÁNÍ UX PRINCIPŮ

CHECKER OF USER EXPERIENCE GUIDELINES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — VÍTĚZSLAV KŘÍŽ
AUTHOR

VEDOUCÍ PRÁCE — Ing. ALEŠ SMRČKA, Ph.D.
SUPERVISOR

BRNO 2015

Abstrakt

Cílem této práce je vytvořit nástroj pro automatizované kontrolování správných vlastností uživatelského rozhraní. Součástí práce je seznam pravidel z prostředí GNOME, KDE a OSX, která jsou vhodná k automatické kontrole. Nástroj na pozadí ověřuje prvky uživatelského rozhraní, zatímco uživatel ovládá testovanou aplikaci. Pravidla jsou ohodnocena podle závažnosti a výstupem programu je mimo jiné i výsledné skóre. V závěru práce jsou otestovány běžně dostupné programy. Výsledná aplikace je vhodná pro ověřování grafického uživatelského rozhraní při vývoji.

Abstract

The aim of this work is to create a tool for automated checking the correct characteristics of the user interface. The thesis includes a list of rules from GNOME, KDE and OSX, which are suitable for automatic control. The tool verifies the background elements of the user interface while the user operates the program under test. The rules are ranked according to the severity and outcome of the program is also overall score. The final part contains testing of normally available programs. The application is appropriate for verifying a graphical user interface in development.

Klíčová slova

testování GUI, UX principy, Dogtail, asistivní technologie, automatizované testování

Keywords

GUI testing, UX guidelines, Dogtail, assistive technology, test automation

Citace

Vítězslav Kříž: Nástroj pro kontrolu dodržování UX principů, bakalářská práce, Brno, FIT VUT v Brně, 2015

Nástroj pro kontrolu dodržování UX principů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doktora Aleše Smrčky. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Vítězslav Kříž
20. května 2015

Poděkování

Tímto bych rád poděkoval doktorovi Alešovi Smrčkovi za odborné vedení mé bakalářské práce.

Obsah

Kapitola 1

Úvod

Nedílnou součástí vývoje softwaru je jeho testování, to probíhá na několika úrovních od jednotkového testování přes integrační testování až po akceptační testování. Pro rychlejší testování aplikací při postupném vývoji probíhá regresní testování, pro které je vhodné mít testy co nejvíce automatizované. Jelikož hodně aplikací má grafické uživatelské rozhraní[1], provádí se část testů právě přes toto rozhraní. Pro podporu automatizace již vzniklo celkem velké množství nástrojů. Tento styl testování je převážně zaměřen na funkčnost aplikace a ovládání grafických prvků slouží pro zadávání příkazů. Typické testování se nezabývá kvalitou uživatelského rozhraní.

Práce se zaměřuje na kontrolu prvků uživatelského rozhraní, nikoliv na vlastnosti celku. Cílem této práce je vytvořit nástroj, který bude automaticky kontrolovat správné vlastnosti uživatelského rozhraní a odhalovat v něm chyby. V následující kapitole je popsán způsob tvorby GUI a také možnost automatizace, neboť na automatizačních nástrojích je založen výsledný program. Všechna pravidla nejsou vhodná k automatizaci. V kapitole 3 je uveden výběr vhodných pravidel s popisem možné automatizace a příkladem použití.

Třídní návrh aplikace je popsán v kapitole 4, implementace je provedena v jazyce Python a s využitím knihovny Dogtail (kapitola 5). V této knihovně byly v průběhu tvorby práce objeveny chyby, jejich popis a řešení naleznete v příloze B. Poznatky z používání výsledného programu jsou popsány v kapitole 6, přičemž program byl využit k otestování běžných aplikací z prostředí GNOME.

[1] GUI – Graphical User Interface

Kapitola 2

Tvorba grafického uživatelského rozhraní

Grafické uživatelské rozhraní slouží pro interakci mezi člověkem a počítačem za použití oken, ikon a menu, je ovládané pomocí myši, ale i klávesnice. Je opakem uživatelského rozhraní pro příkazovou řádku, které používá pouze text a klávesnici.[1]

2.1 Uživatelské rozhraní pro pracovní plochu

Pro vytváření prvků uživatelského rozhraní se používají dostupné nástroje operačního systému, externí programové knihovny nebo kombinace obojího. Použití takovýchto nástrojů zajistí víceméně jednotný vzhled aplikace s ostatními programy operačního systému. Méně časté je tvoření uživatelských prvků pouze pomocí základních funkcí určených pro vykreslování, které však za cenu náročnější práce umožňují vytvořit nestandardní a daleko rozmanitější vzhled aplikace.

Existuje celá řada nástrojů a knihoven, které se používají na různých platformách. Hodně těchto knihoven je multiplatformních, a to dokonce i tak, že se používá stejný nástroj jak pro osobní počítač, tak pro mobilní zařízení (např. Qt).

I přes některé rozdíly mezi jednotlivými systémy je základní princip tvorby uživatelského rozhraní podobný. Prvky rozhraní (widgets) jsou reprezentovány objekty tak, jak je známe z objektového programovacího paradigmatu. Tyto objekty se skládají do stromové struktury, která poté reprezentuje uživatelské rozhraní celé aplikace. Vytvoření této stromové struktury objektů může být dosaženo dvěma způsoby.

Prvním způsobem je vytváření jednotlivých objektů přímo zápisem ve zdrojovém kódu, definovat jejich parametry a závislosti. Tento způsob umožňuje vytvářet rozhraní dynamicky na základě dat z programu, což může znepřehlednit návrh aplikace, avšak vytváření uživatelského rozhraní je náročnější na představivost a znalosti.

Pro zjednodušení vývoje se využívá aplikací, které umožňují vytvářet uživatelské rozhraní pomocí umisťování prvků do okna. Tyto editory patří do skupin programů označovaných jako WYSIWYG[1].[6] To umožňuje vytvářet prototypy rozhraní velice rychle a relativně nezávisle na funkcionalitě samotného programu, designér uživatelského rozhraní tedy nemusí mít programátorské znalosti. [7] Vytvořená struktura objektů je následně uložena do souboru v jazyce založeném na XML[2].[15]

[1] What you see is what you get

[2] XML – Extensible Markup Language je obecný značkovací jazyk. Používá se pro serializaci dat.

2.1.1 Gtk+

Gtk+ nebo taky GIMP Toolkit je multiplatformní nástroj pro tvorbu grafických uživatelských rozhraní. Toolkit je napsán v jazyce C, ale je použitelný i v jiných programovacích jazycích. [17]. Prvky uživatelského rozhraní jsou reprezentovány objekty, které může programátor vytvářet přímo v programovacím jazyce nebo může strukturu objektů zapsat do XML souboru. Pro editaci tohoto souboru je vhodné použít nástroj Glade, jenž umožňuje vytvářet uživatelské rozhraní i bez znalosti programování. [15]

2.1.2 Windows Presentation Foundation

Windows Presentation Foundation je součástí NET Frameworku od verze 3.0. Nahradil starší, ale doteď ještě používanou knihovnu Windows Forms.

Pro zapisování objektů uživatelského rozhraní se používá jazyk XAML[3]. Tento jazyk není závislý na programovacím jazyce[4], který se používá pro psaní programu. Zápis v XAML je zkompilován do binární podoby a při spuštění programu jsou za běhu vytvořeny definované objekty.

Nástroje integrované ve Visual Studiu umožňují vytvářet uživatelské rozhraní přímo zapisováním v XAMLu, přičemž je ihned vidět výsledek. Případně jde tvořit obsah okna umisťováním jednotlivých prvků nebo také zkombinováním obou způsobů. Pro složitější návrh je vhodné využít program Blend. [18]

2.2 Webové rozhraní

Webové technologie sloužily původně jako statické či dynamické prezentace. V poslední době se díky technologiím jako HTML5[13], Javascript, AJAX rozšiřují webové aplikace, které jsou ekvivalentem aplikací desktopových (např. Google Docs).

Pro tvorbu webových aplikací se používají obdobné principy jako u aplikací desktopových. Existuje velké množství knihoven poskytujících widgety[5], jako příklad bych uvedl jQWidgets[10] a Kendo[14] . Společným prvkem těchto knihoven je výstup ve značkovacím jazyce HTML s doplněním CSS a Javascriptu. Prvky uživatelského rozhraní mohou ztrácet sémantickou informaci, což způsobuje problém asistivním technologiím.[3]

2.3 Nástroje pro testováni GUI

Důležitou součástí vývoje programů je testování. V případech, kdy má výsledná aplikace GUI, je běžné, že minimálně část testování probíhá prostřednictvím tohoto grafického rozhraní. Pro automatické testování slouží různé nástroje. V této kapitole bych rád popsal některé z nich.

2.3.1 Sikuli

Sikuli script je nástroj, který dokáže automatizovat všechno, co jde vidět na obrazovce.[5] Používá rozpoznávání obrazu a textu pro identifikování a ovládaní prvků uživatelského

[3] XAML – Extensible Application Markup Language je jazyk založený na XML. Slouží k deklarativnímu vytváření objektů a nastavování jejich vlastností. Ačkoliv se využívá ve WPF pro tvorbu prvků UI, je možné jej využít obecně pro vytváření i jiných objektů

[4] Jazyk musí být kompatibilní s platformou .NET.

[5] Widget – prvek uživatelského rozhraní

rozhraní. To tento nástroj předurčuje k ovládání uživatelského rozhraní, pro které nemáme jiné možnosti nebo například nemáme přístup ke zdrojovému kódu. Velkou výhodou tohoto nástroje je jeho multiplatformnost. Do základní knihovny je přidávána pouze funkcionalita, která je dostupná zároveň na systémech Windows, Linux a Mac.[8] Pro podporu testování umožňuje kontrolovat přítomnost, či právě nepřítomnost určitého prvku.

Pro automatizované ověřování principů uživatelského rozhraní je tento přístup méně vhodný, protože je zaměřen primárně na ovládání. Jenže pro ověřování je nejvíce důležité pozorování. Rozpoznávání obrazu je pomalé a závisí na výkonu grafické karty.

2.3.2 Selenium

Selenium je nástroj pro automatizování webového prohlížeče. Součástí projektu je několik nástrojů, které umožňují i vzdálené ovládání několika prohlížečů zároveň. Pracuje nad DOM[6] modelem webové stránky, pro identifikaci elementů používá jazyk XPath[7]. Používat Selenium Web Driver je možné z jazyka Java, C#, Ruby, Python nebo Javascript.[2] Pro testování webových stránek je Selenium velmi vhodným nástrojem.

2.3.3 Dogtail

Dogtail je automatizační nástroj, který využívá asistivní technologie pro získávání informací o prvcích uživatelského rozhraní a pro jejich kontrolování. Je napsán v jazyce Python a z velké části využívá knihovnu AT-SPI a její portaci pro Python. Tento nástroj je funkční v prostředí GNOME a KDE.

Asistivní technologie, označované také zkratkou a11y, jsou součástí počítačů a operačních systému i na základě některých národních zákonů a norem. Mezi pomocné nástroje patří obrazovková lupa, předčítání textu, alternativní ovládání kurzoru nebo klávesnice na obrazovce. Pro tyto nástroje jsou v operačních systémech připraveny podpůrné knihovny. Tyto nástroje tak mají přímý přístup k ovládání všech aplikací, získávání informací z nich a zachytávání událostí ze vstupních zařízení. To představuje určité bezpečnostní riziko a jejich použití musí uživatel explicitně povolit. [9] Protože Dogtail využívá přímo informace o prvcích uživatelského rozhraní, je ovládání aplikace velice rychlé. Na rozdíl od Sikuli však dokáže automatizovat pouze některé aplikace.

[6] DOM – Document Object Model je objektová reprezentace XML nebo HTML dokumentu. Poskytuje rozhraní pro modifikaci obsahu a struktury dokumentu.

[7] XML Path Language – Je jazyk pro adresování části XML dokumentu. Pomocí tohoto jazyka je možné vybírat jednotlivé elementy a atributy.

Kapitola 3

Pravidla

UX princip je výrok navrhující doporučení a uvážení o návrhu specifického jevu nebo komponenty v určitém kontextu. Některá návrhová pravidla vycházejí ze studií, ale většina vychází z principů, zásad a zkušeností. [7]

Pro běžné rozšířené grafické prostředí existují návody a pravidla, jak používat konkrétní prvky uživatelského rozhraní, ale hlavně jak je nepoužívat. V této kapitole jsou popsána pravidla vybraná z dokumentů pro prostředí GNOME[16], KDE[11] a OSX[4], která mají potenciál pro automatizované ověřování.

Pravidla by se měla aplikovat rozumně vždy v určitém kontextu, pokud je nějaké pravidlo porušeno, nemusí to znamenat nutně chybu, nýbrž nejlepší kompromisní řešení. Například ve Visual Studiu, vývojovém prostředí od Microsoftu, které je velmi rozsáhlé, mohou být určitě porušeny některé zásady pro vývoj programů v tomto prostředí. [12]

V této kapitole popisuji pouze pravidla, která mají určitý potenciál pro automatizovanou kontrolu.

3.1 Společná pravidla

Společná pravidla, která platí obecně pro více prvků. Na tato pravidla je v textu dále odkazováno.

3.1.1 Psaní malých a velkých písmen

Začněme psaním malých a velkých písmen. Tato pravidla jsou velmi zajímavá, při jejich automatické kontrole by mohlo dojít k neúspěchu. Problémy nacházíme zejména s rozpoznáváním sloves, vyhledáváním slov ve slovnících a existencí určitých výjimek. Otázkou je, zda se dají aplikovat na jiný jazyk než angličtinu, z tohoto důvodu bude v následujících příkladech použito anglických výrazů.

Header Capitalization

Pravidlo *Header Capitalization* se používá v GNOME pro nadpisy, položky menu a tlačítka. První písmeno se píše velké u:

- všech slov delších než 4 znaky,
- sloves jakékoliv délky,

- prvního a posledního slova,
- složených slov se spojovníkem (např. Self-Test).

V prostředí KDE je uplatňováno obdobné pravidlo s názvem *Title Capitalization*.

Sentence Capitalization

Toto pravidlo je aplikováno na *Checkbox*, *Radio button*, *Slider* nebo popisek u textového pole. První písmeno se píše velké u prvního slova a dále u slov, která se běžně píšou s velkým písmenem.

3.1.2 Výpustka

Pokud se nacházíme v situaci, kdy tlačítko nebo položka menu nevyvolá přímo konkrétní akci, ale vyžaduje zadání dalších informací, tak by měl text takového prvku končit třemi tečkami. Typickou ukázkou je volba „Tisk . . . “ v hlavním menu, kdy trojtečka nám dává dopředu vědět, že se samotný tisk ještě neprovede.

Dále bych se chtěl zmínit o možné realizaci automatické kontroly. Pokud se po kliknutí na prvek zobrazí nové okno nebo dialog, proběhne zpětná kontrola, zda má tlačítko trojtečku. Lze kontrolovat i opačný jev, kdy má prvek trojtečku, ale nevyžaduje žádnou další akci.

Může nastat i situace, kdy se jedná o výjimku. Řadíme zde položky jako Nastavení, Vlastnosti, Volby, z jejichž významu je patrné, že se otevře další okno.

3.1.3 Access Key

Access Key umožňuje uživateli ovládat prvky s popiskem pomocí klávesové zkratky s tlačítkem Alt. To je naznačeno pomocí podtržení písmena v popisku. Většinou až po stisku Alt je podtrženo písmeno, jehož stisknutím vyvoláme akci na daném prvku. V různých nástrojích na tvorbu rozhraní toho můžeme dosáhnout napsáním podtržítka před dané písmeno. Jedná se o velmi jednoduše testovatelnou vlastnost. Pokud je to možné, všechny prvky s popiskem by to měly mít přiřazeno.

Důležité je zmínit, že nesmí být dvě stejné zkratky zároveň přístupné. Jako podtržené písmeno je nevhodné použít úzké znaky (malé i nebo l) nebo znaky zasahující pod řádek (g, y, j), protože toto podtržení není s těmito znaky tak zřetelné.

3.2 Tlačítko

Účelem tlačítka je vyvolání určité akce. Hlavní podmínkou je, že akce by měla být vyvolána pouze po kliknutí levým tlačítkem myši. Rozhodně by nemělo být aktivní na dvoj-klik či dokonce pravý klik. Uživatelé takovou akci zřídkakdy objeví, a pokud ano, budou jejich očekávání ohledně dalších tlačítek narušena.

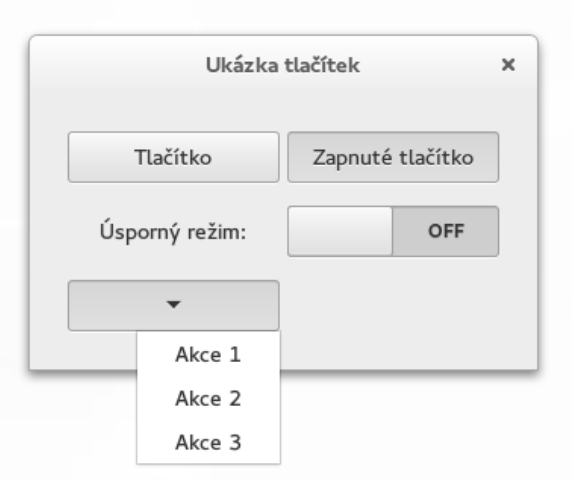
Co se týče pravidel pro obsah textu tlačítek, platí pro ně ta výše zmíněná, tedy přesněji *Header Capitalization*, *Ellipses* a *Access Key*. Správně by v textu tlačítek mělo být sloveso, je to však automaticky nekontrolovatelné.

3.2.1 Podtypy

Tlačítka nemají pouze jednu funkci, lze je použít i jako jiný typ prvku. Uveďme například *Toggle Buttons* (přepínací tlačítka), jenž používáme místo *Checkboxu* a zaškrtnuté tlačítko vypadá, jako kdyby bylo neustále stisknuté. KDE má zde pro tento typ speciální pravidlo, naopak OSX se tomuto striktně vyhýbá. Aktivace *Toogle Buttonu* nesmí vyvolat žádnou akci, jako je běžné u normálního tlačítka. Titulek se nesmí měnit v závislosti na stavu tlačítka, pro oba stavy musí být stejný. Existuje také typ *Menu Button*, kdy se po jeho kliknutí zobrazí menu, avšak OSX to nedoporučuje. Odlišujícím prvkem je zobrazení šipky dolů za textem, která naznačuje zobrazení menu.

3.2.2 Rozměry

Všechna tlačítka v jednom okně by měla mít konzistentní rozměry. Důležité je, aby tlačítka vedle sebe dosahovala stejných rozměrů. Není dobré používat více než jednu či dvě rozdílné šířky tlačítek v jednom okně, měla by být o stejné výšce. Dosáhneme tak nejen příjemných jednotných vizuálních výsledků, ale usnadní to i užívání. Pokud je více tlačítek umístěno vedle sebe, musí být stejně široká, a to zejména u párů tlačítek OK a CANCEL.



Obrázek 3.1: Ukázka různých typů tlačítek

3.2.3 Výchozí tlačítko

Dialog by měl mít výchozí tlačítko, které se zobrazuje jiným stylem a jehož aktivace se provádí klávesou Enter, avšak toto pravidlo nelze striktně vyžadovat kvůli výjimce pro dialogy jako Odstranit nebo Zavřít bez uložení, kde by při použití výchozího tlačítka nastala nevratná destruktivní akce. Je zde možnost automaticky upozornit na nepřítomnost výchozího tlačítka u všech dialogů, tester poté musí sám podle svého úmyslu rozhodnout, zda se jedná v onom konkrétním případě o chybu nebo výjimku.

3.2.4 Vyvolání akce

Jak již bylo zmíněno, primární funkcí tlačítka je vyvolání akce. Tato akce by měla být pro uživatele viditelná a musí nastat bezprostředně po stisknutí, případně musí být uživatel informován o tom, že probíhá určitá činnost na pozadí. Toto pravidlo jde automaticky kontrolovat, přínos kontroly je ale pouze v případě, že kontrola bude probíhat plně automaticky bez asistence uživatele, v opačném případě je to naprosto zbytečné, jelikož uživatel jev rozpozná daleko rychleji a lépe.

Pokud akce není v momentálním kontextu proveditelná, je vhodnější mít tlačítko deaktivované (zobrazené šedě, bez možnosti kliknutí), než aby se místo akce zobrazil chybový dialog. Tento jev však nemohu označit chybným, jelikož ne vždy je možno ihned vědět, že akce je proveditelná. Pro automatické ověření je důležité rozpoznat chybový dialog. Dále

stojí za zmínku, že při vyvolání jiného dialogu je nutné uplatnit pravidlo o výpustce (kapitola 3.1.2).

3.2.5 Umístění

Tlačítko se stejnou funkcí by mělo být ve všech oknech na přibližně stejné pozici. Funkce tlačítka bohužel nejde automaticky rozpoznat, jedině podle popisku tlačítka. Testování tohoto jevu by bylo obtížnější. Testuje se vztah mezi více okny, musí se ukládat všechny pozice a všechno porovnávat.

3.3 Checkbox

Hlavní funkce *Checkboxu*[1] je vyjádření dvou možností – zapnuto, vypnuto. Existuje i třetí stav označovaný *mixed*. Pokud je opak slova nejasný, je lepší použít dva *Radio Button* vedle sebe. Například pro volbu otočení stránky „Na výšku“, „Na šířku“ nelze použít pouze *Checkbox*, protože opačný význam není zřejmý. Automatizovaně nelze rozhodnout, zda je lepší použít *Checkbox* nebo dva *Radio Buttony*. Správné použití je i v dialogu pro nastavení času (obrázek 3.2), kde je výběr mezi možností Analog a Digital.

Každý *Checkbox* by měl mít svůj popisek (label), který při kliku funguje stejně jako samotný čtvereček na zaškrtnutí, a také přiřazenou klávesu pro rychlý přístup. *Checkbox* vytvořený pomocí systémových knihoven většinou přímo obsahuje popisek, který je jeho součástí. Pokud by *Checkbox* název neměl, je možné, že bude mít popisek umístěn nezávisle vedle sebe. V takovém případě je třeba zkontrolovat, jestli klik na popisek vyvolá stejnou akci. Více *Checkboxů* je vhodné organizovat do skupin, přičemž skupina by měla mít vhodný nadpis. Pro automatickou kontrolu je potřeba určit, co je skupina *Checkboxů* a taky způsob, jak najít nadpis. Správné rozložení skupin je na obrázku 3.2.

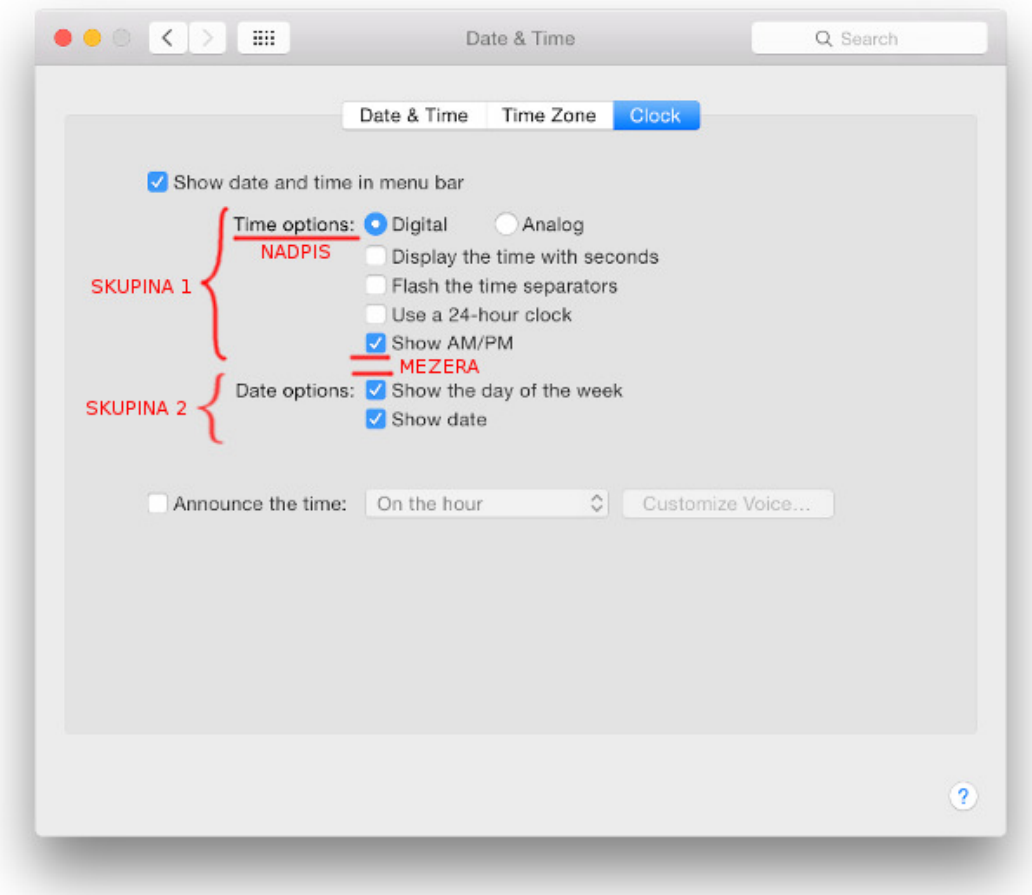
Použití stavu *mixed* je možné pouze v případě, když má *Checkbox* další potomky organizované do stromové struktury. Tento stav by neměl jít uživatelem zvolit přímo, ale pouze znamená, že nejsou všichni potomci ve stejném stavu. Pro správnou automatickou kontrolu je potřeba z vizuálního rozložení určit hierarchii *Checkboxů*. Za zmínění stojí pravidlo z GNOME, které říká, že pokud se proklikáme přes všechny stavy zpět na *mixed*, mělo by se vrátit původní nastavení. Oponentem GNOMU je KDE, zakazuje totiž přímé nastavení stavu *mixed*.

Aktivace nebo deaktivace *Checkboxu* by neměla měnit žádné hodnoty jiných prvků. Může maximálně zobrazovat nebo povolovat jiné prvky. Co se týče polohy, *Checkboxy* by měly být zarovnány ve sloupci pod sebou. Počet prvků pod sebou by měl být omezen (KDE max. 5, GNOME max. 8), z toho důvodu by bylo vhodné rozpoznávat podle umístění skupinu a nadpis, typické rozmístění je na obrázku 3.2.

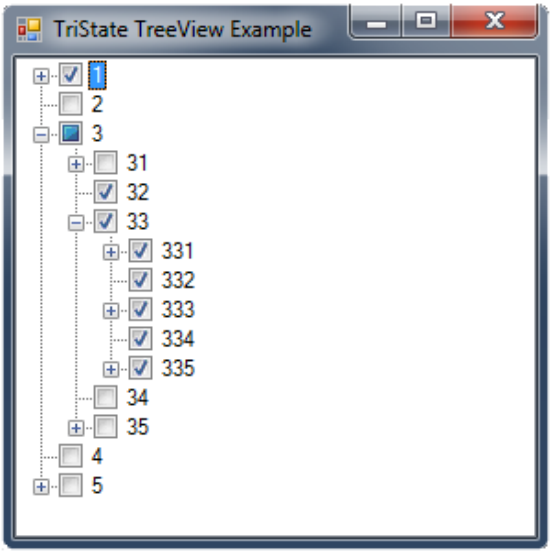
3.4 Radio Button

Radiobutton, v překladu by se snad dalo použít výrazu přepínací tlačítko, má některá pravidla podobná s *Checkboxem*. *Radio Button* má smysl pouze při výskytu ve skupině, kdy je vybrán právě jeden prvek. Pokud by bylo potřeba za validní vstup považovat i nevybrání žádného prvku, používá se pro tuto možnost další speciální prvek značící nevybrání žádné možnosti.

[1] Český překlad zatrhávací pole nebudu v textu používat.



Obrázek 3.2: Ukázka správného rozložení skupin i nahrazení *Checkboxu* pomocí dvou *Radio Buttonů*.



Obrázek 3.3: Správné použití třetího (neurčitého) stavu u *Checkboxu*. Zdroj: http://www.codeproject.com/Articles/202435/Tri-State-Tree-View

Prvky, které jsou funkčně propojeny ve skupině, musí být i vizuálně sdruženy do skupiny. Je vhodné, aby skupina měla nadpis. Automaticky lze testovat skupiny prvků na základě změny jejich stavů. Prvky, jež se mění společně, patří do jedné skupiny.

Co se týče povolených akcí, platí stejná pravidla jako pro *Checkbox*. *Radio Button* neslouží k vyvolání akce, nemá měnit hodnoty jiných prvků a může skrývat nebo zobrazovat jiné prvky. Pravidla jsou totožná i pro umístění, tedy zarovnat do sloupců pod sebe a omezit množství prvků.

3.5 Jednořádkové textové pole

Jednořádkové textové pole má v různých prostředích názvy: TextField – GNOME, LineEdit – KDE, TextInputField – OSX, TextBox – Windows.

Tento prvek slouží pro zapsání jednoho řádku a jeho použití je omezeno pouze na krátké texty. Popisek není součástí prvku, na rozdíl od *Checkboxu* je jej nutno přidat. Popisek by se měl nacházet vlevo a měl by mít přiřazen *Access Key*.

Rozměry by měly být takové, aby byl celý text pořád viditelný. To poskytuje uživateli užitečný vizuální podnět o množství očekávaného textu a zabraňuje přetékání. Zde by bylo vhodné upozornit na situaci, když se text nevejde. Otázkou je, zda půjde zjistit, že je text moc dlouhý.

Při výskytu více prvků v okně jsou žádoucí dostatečné mezery mezi jednotlivými vstupními prvky na jednom řádku, jejich zarovnání pokud jsou pod sebou a stejná délka v jednom okně.

Mluvíme-li o kontrole reakce na vstup, bylo by možné kontrolovat, že to, co uživatel opravdu píše, se objevuje v editačním poli. Tím by se ale pouze kontrolovala základní funkčnost. A pokud by se opravdu něco nenapsalo, tak to musel způsobit program záměrně, například pro omezení vstupních znaků, maximální délku vstupu, špatný formát. Pokud program zabraňuje napsání znaku, musí o tom uživatele informovat. Nepodaří-li se najít žádný způsob informování, je třeba upozornit na možnou chybu.

3.6 Víceřádkové textové pole

Víceřádkové textové pole je využíváno pro dlouhý text, případně zapsání více řádků. V prostředí GNOME se používá stejný prvek jako pro jednořádkové pole, pouze se změní parametry. Neměl by se vyskytovat horizontální posuvník (*Scrollbar*) a velikost by měla odpovídat očekávanému textu.

3.7 Slider a Spinbox

Jedná se o dva rozdílné prvky uživatelského rozhraní, které slouží převážně pro výběr číselné hodnoty.

3.7.1 Spinbox

Spinbox (v OSX *Stepper*) je prvek podobný text boxu sloužící pro zadávání číselných hodnot. Upravit hodnotu je možné psaním na klávesnici nebo pomocí dvou tlačítek, zpravidla umístěných vlevo a označených šipkami nahoru a dolů (případně plus a minus). Prvek se má používat pouze pro číselné údaje i přesto, že je technicky možné použít tento prvek pro

výběr nečíselných hodnot, pro které je správné použít *Listbox* (používá se ve vzoru *Dual List* na obrázku 3.7), *Drop Down List* nebo *Combobox*3.11.

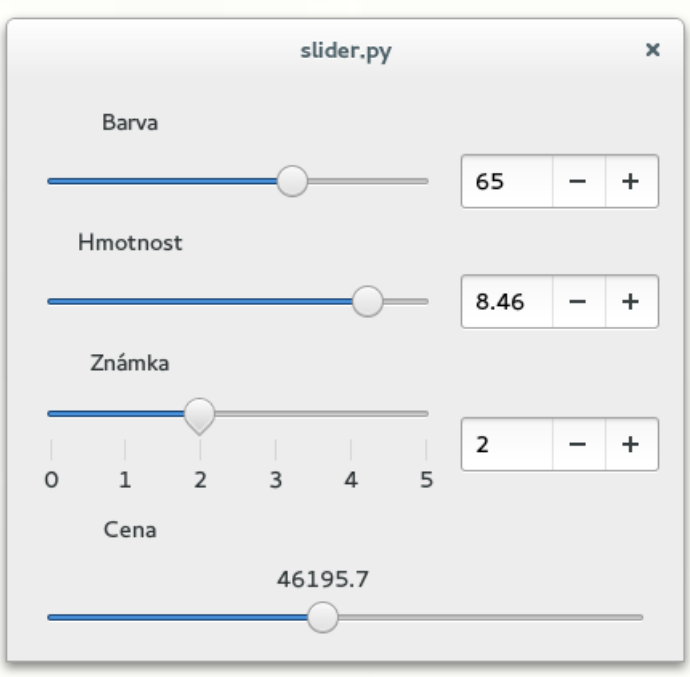
3.7.2 Slider

Posuvník slouží pro zadání hodnoty z vymezeného rozsahu, například míra zvětšení dokumentu, hlasitost hudby atd. Druhou možností je využití posuvníku jako vyhledávací pruh v přehrávači hudby, v tomto případě nemusí být označen popiskem.

3.7.3 Kombinace

Spinbox a *Slider* je někdy vhodné propojit dohromady. V takovém případě slouží *Spinbox* pro zobrazení hodnoty a pro přesné zadávaní, kdežto *Slider* umožňuje provádět rychlejší změnu přes velký rozsah hodnot. Provázané prvky se musí ihned měnit při změně druhého prvku.

Pouze samotný *Spinbox* se použije za situace, kdy rozsah zadávaných čísel je z jedné nebo obou stran neomezený, v tomto případě je nemožné využít *Slider*. Pokud je rozsah oboustranně omezený, je vhodné použít *Slider*. Kombinace obou prvků je vhodná, pokud je rozsah omezený a počet možných hodnot je větší než 20, nebo lze použít desetinné číslo. Při kombinaci slouží *Slider* pro rychlou změnu hodnoty a *Spinbox* pro zadání přesného údaje.



Obrázek 3.4: První dva příklady jsou ukázkou správného použití, druhé dva nevhodného. U položky Známka stačí pouze *Slider*, protože se vybírá jen ze šesti hodnot, naopak u položky Cena chybí *Spinbox*, jelikož můžeme zadávat čísla z velkého rozsahu.

3.8 Panel nástrojů

Panel nástrojů (*Toolbar*) je pás ovládacích prvků, které poskytují pohodlný a rychlý přístup k často používaným funkcím v aplikaci. Zpravidla jsou tlačítka označena pouze ikonkou, na jejíž vzhled by měl být brán zřetel, vzhled ikonek je také specifikován pravidly, která zde ale nejsou zmíněna.

Automaticky kontrolovatelná vlastnost znamená, že pokud je v aplikaci použito zároveň i hlavní menu, pak všechny akce přístupné přes panel nástrojů musí být dostupné i z menu. Po kliknutí na prvek *Toolbaru* by měla být ihned vykonána akce, není možné žádat další vstup od uživatele jako v případě výpustky 3.1.2.

KDE specifikuje rozdělení prvků v *Toolbaru* do maximálně tří skupin s maximálním počtem 5 prvků. Tento limit se zdá být velice přísný a komplexnější programy jej určitě překonají.

3.9 Menu

Rozlišujeme hlavní menu v okně aplikace, které obsahuje další menu, dále pak ještě kontextové menu. Pro položky menu platí stejná pravidla.

Hlavní menu se zobrazuje jako pruh umístěný nahoře v aplikaci těsně pod okrajem primárního okna[2] aplikace. V horním menu musí být titulky jednoslovné, dvouslovný název by zmátl uživatele, který by mohl považovat každé slovo zvlášť jako samostatné menu. Samozřejmostí je použití *Access Key* (kapitola 3.1.3) a psaní první písmeno velkým písmenem (kapitola 3.1.1).

V prostředí GNOME a OSX má každá aplikace navíc i aplikační menu (viz. obrázek 3.5). Položky v tomto menu jsou na stejné logické úrovni jako položky v hlavním menu v okně aplikace.

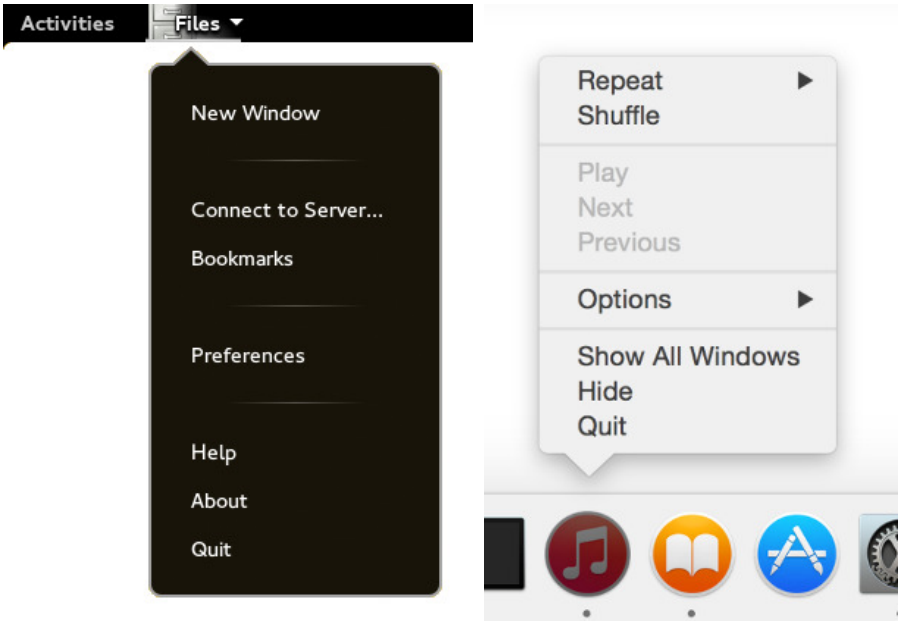
V hlavním menu má být maximálně 9 dalších menu položek, jelikož větší počet je zatěžující a znesnadňuje používání. I zde se vyskytují podmínky, které by se měly dodržovat. Za prvé nedeaktivovat položky menu, které zobrazují další menu. Dále by neměla existovat možnost skrytí menu. V Mozille (Firefox, Thunderbird) je to běžná věc a menu je ve výchozím nastavení skryté, nicméně přínos tohoto testování je mizivý. Prvky v hlavním menu by měly být statické, to znamená nepřidávat nebo neubírat prvky. Pokud je nějaká položka v aktuálním kontextu nepoužitelná, tak ji lze učinit pouze neaktivní.

Počet prvků v každém menu by měl být mezi třemi a dvanácti, submenu maximálně šest. Pro kontextové menu specifikuje KDE ještě přísnější limit 10 prvků. Je vhodné organizovat menu do skupin pomocí oddělovačů. V jedné skupině má být pouze jeden typ položky[3], přičemž skupina může obsahovat maximálně sedm položek. Menu může obsahovat další submenu, přičemž je přípustná pouze jedna úroveň zanoření.

Menu může obsahovat další submenu, přičemž maximální úroveň zanoření je 1. Pro hlavní menu aplikace to znamená například Soubor » Odeslat » Něco. Více zanořené menu je pro uživatele již nekomfortní.

[2] Primární okno je první zobrazené okno po startu aplikace. Rozlišujeme jedno-instanční aplikace (například emailový klient), u kterých lze otevřít pouze jedno primární okno a více-instanční (například různé editory), u kterých je běžné zobrazit více stejných primárních oken s různým obsahem.

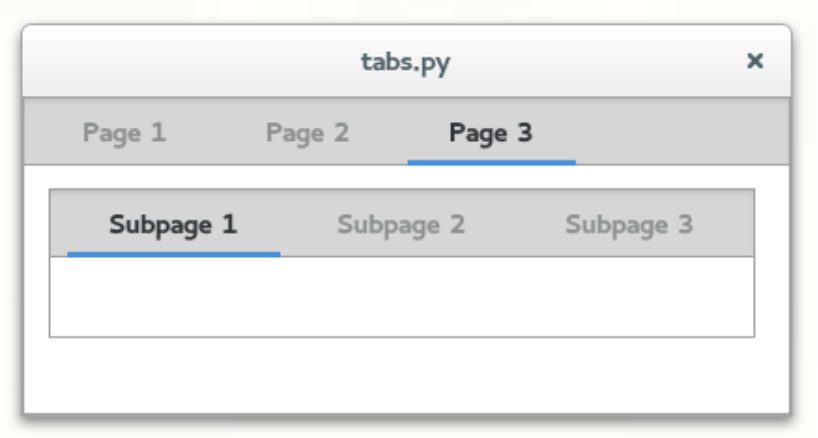
[3] V menu obecně můžou být položky reprezentující *Checkbox*, *Radio Button* nebo vyvolání akce (tlačítko)



Obrázek 3.5: Vlevo Application Menu z prostředí GNOME, vpravo Dock Menu ze systému OSX.

3.10 Záložky

Záložky můžeme rozdělit na dvě varianty, dynamické a statické. Statické se obvykle používají v nastavení, oproti tomu dynamické v geditu pro jednotlivé otevřené soubory nebo ve firefoxu pro jednotlivé stránky, lze je přesouvat i zavírat. Automatizovaně se dají od sebe rozpoznat podle chování. Pokud se počet záložek změní, jsou dynamické, do té doby je můžeme považovat za statické.



Obrázek 3.6:

Změna na jednom panelu statických záložek by neměla ovlivňovat jiný panel. Šířka „ucha“ by měla přesně odpovídat textu u statických záložek, u dynamických je běžné použít stejnou šířku. Záložky musí být stále přístupné, což znamená, že deaktivovaný tab je automaticky chyba. Experimentováním v různých nástrojích a prostředích jsem zjistil, že

pomocí standardních prostředků nelze dosáhnout neaktivní záložky, proto nemá smysl se zaměřovat na tuto kontrolu. V žádném případě se nesmí jednotlivé záložky zanořovat (jak je ukázáno na obrázku 3.6).

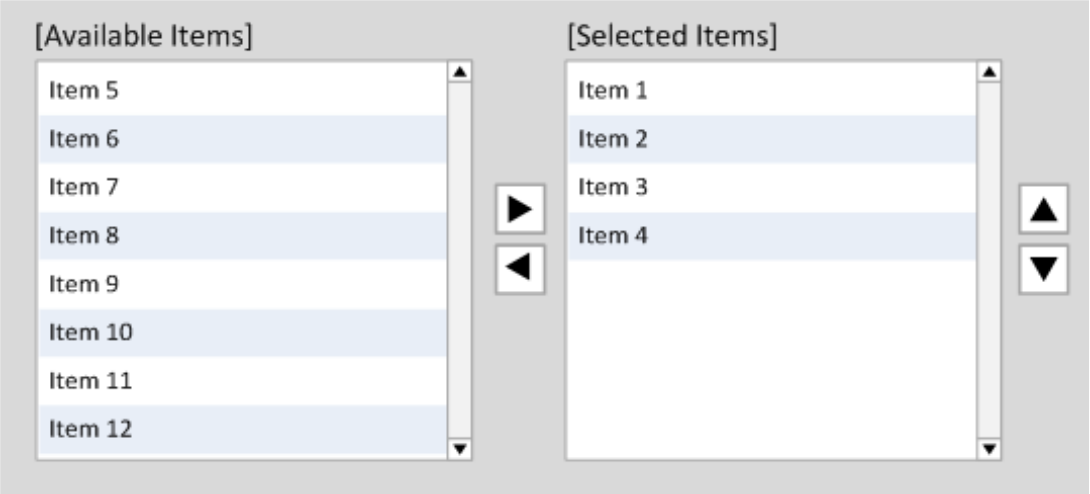
3.11 ComboBox

Výsuvný seznam nebo také DropdownList umožňuje výběr jedné z možností. Nahrazuje skupinu *Radio Buttonů* v případě, že je potřeba vybírat z mnoha možností nebo v okně není dostatek místa. Může se využít i možnosti přidání nové, uživatelem zadané, položky do seznamu. Jako každý jiný vstupní prvek musí disponovat i *Combobox* popiskem s *Access Key*. Stejně jako u *Radio Buttonu* je důležité mít vždy vybraný právě jeden prvek a také je zde možno použít speciální volbu znamenající, že nebylo nic vybráno.

Stejně jako *Checkbox*, když například mění jiný prvek, měl by být tento prvek umístěn v blízkosti. Vzhledem k *Checkboxu* můžeme pozorovat ten rozdíl, že ComboBox by prvky neměl skrývat, ale pouze je deaktivovat.

3.12 Dual list

Nejedná se o jeden prvek, nýbrž o vzor složený z více prvků. Rozeznáme dva seznamy vedle sebe, levý obsahuje všechny položky, pravý vybrané. Mezi seznamy jsou tlačítka s ikonami vlevo a vpravo pro přesouvání položek. Tato tlačítka musí být aktivní podle aktuální situace. Dále musí fungovat dvojklik a funkce drag and drop. Pokud se podaří rozpoznat některé znaky tohoto vzoru, dalo by se upozornit na chybějící funkcionalitu.



Obrázek 3.7: Ukázka vzoru DualList. Zdroj: https://techbase.kde.org/Projects/Usability/HIG/DualList

Kapitola 4

Návrh aplikace

V této kapitole je zmíněn návrh základního principu fungování aplikace, obsahuje také návrh tříd a uživatelského rozhraní.

4.1 Způsob testování

Pro ověřování principů GUI se nabízí dva poněkud odlišné přístupy.

Nástroj si sám spustí testovaný program, bude provádět akce místo uživatele a přitom aplikovat pravidla pro kontrolu. U takového přístupu nevadí, když testování bude pomalé, neboť není vyžadována účast člověka. Algoritmy pro kontrolu pravidel mohou ovládat prvky rozhraní tak, aby ověřily co nejvíce možností. Toto řešení však přináší několik problémů. Musí se určit, na které prvky se bude klikat a v jakém pořadí. Samozřejmě je možné ovládat aplikaci náhodně. Při tom by však mohlo dojít k některým nenávratným změnám (například smazání souboru z disku), je také docela možné, že by se aplikace dostala do chybného stavu. Nejmenším problémem by bylo samotné ukončení aplikace nebo testovacího nástroje.

Druhou možností je spustit nástroj pro testování na pozadí a uživatel (programátor nebo tester) bude sám ovládat testovanou aplikaci, přičemž se bude provádět automatická kontrola. Tato kontrola bude navázána na události uživatelského prostředí a události vstupních zařízení (myš a klávesnice). Aplikování pravidel musí být rychlé, nesmí zdržovat samotné ovládání programu. To téměř diskvalifikuje rozpoznávání obrazu pro kontrolu pravidel. Tento způsob jsem zvolil pro návrh a implementaci nástroje.

4.2 Možnost ignorování

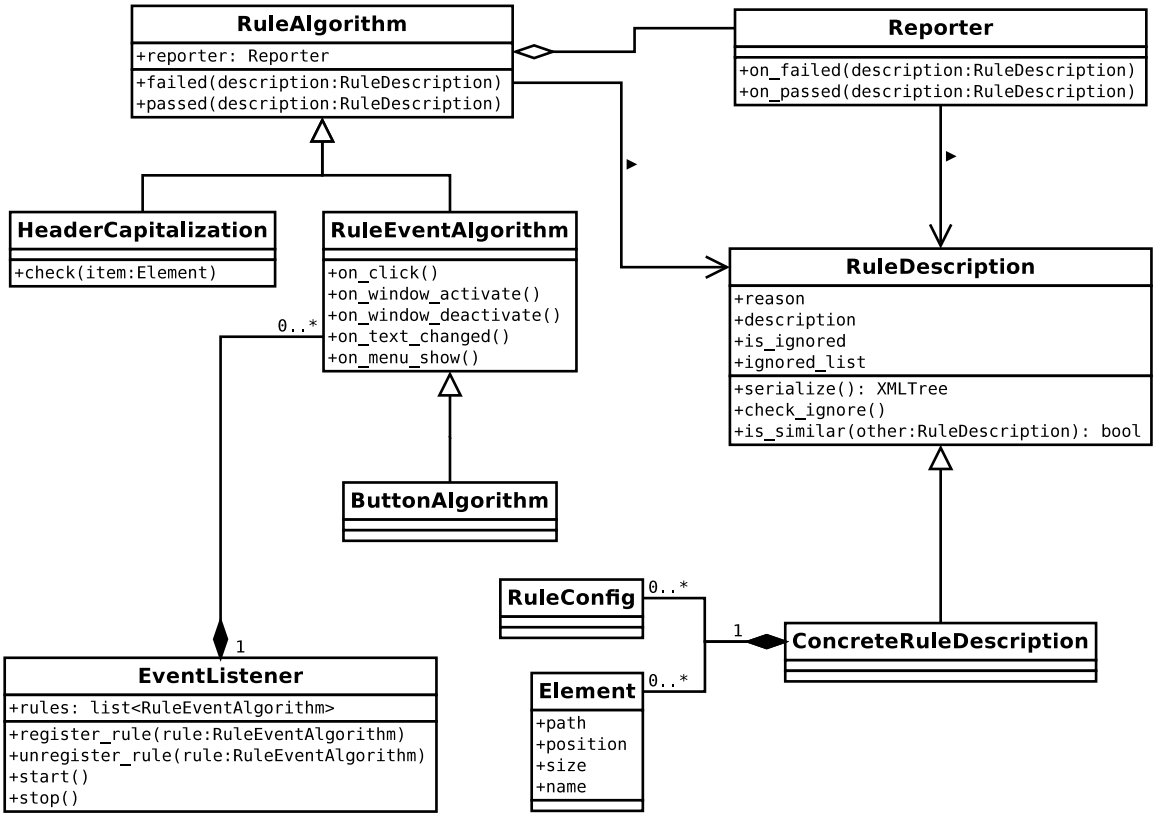
Jelikož hodně GUI principů používá výrazy jako „mělo by být“ nebo „bylo by vhodné“ a nejsou tudíž tak striktní, je třeba zavést možnost ignorování pravidla pro určitý prvek rozhraní. I při kontrole pravidel správného psaní zdrojového kódu programu je většinou dána možnost pravidlo pro určitý jev ignorovat, kde toto napíšeme přímo před tu část kódu, ve které chceme pravidlo vypnout. V grafickém uživatelském prostředí to není možné, neboť jej nemůžeme změnit ani k tlačítku dopsat komentář.

Pro prvky uživatelského rozhraní je nutno použít jednoznačnou identifikaci v rámci aplikace nebo aktuálního okna. Protože jsou jednotlivé elementy uspořádány do stromové struktury, bylo by vhodné použít identifikaci založenou na jazyce XPath, který používá i nástroj Selenium (zmíněno v kapitole 2.3.2).

4.3 Návrh tříd

Třída `EventListener` bude zpracovávat události z knihovny AT-SPI, pro každou zpracovanou událost zavolá příslušnou metodu ve všech objektech třídy `RuleEventAlgorithm`, které jsou pro odběr události zaregistrované. Algoritmy pro kontrolu pravidel uživatelského rozhraní jsou v samostatných třídách, které dědí od `RuleAlgorithm` nebo `RuleEventAlgorithm`. V obrázku 4.1 je třída `HeaderCapitalization` příkladem pravidla, které musí být voláno explicitně z jiného pravidla, kdežto `ButtonAlgorithm` je volán na základě události z `EventListener`.

Od třídy `RuleDescription` dědí všechny třídy reprezentující určité pravidlo. Pokud je kontrolován jev v algoritmu, vytvoří se odpovídající objekt podtřídy `RuleDescription` i s informací, kterého konkrétního prvku uživatelského rozhraní se týká. Takto vytvořený objekt je předán metodě `failed` nebo `passed` třídy `RuleAlgorithm`.



Obrázek 4.1: UML diagram tříd

Z důvodu potřeby uchovávání zaznamenaných chyb a jejich následnému zpracování k výsledku je vhodné použít samostatnou třídu `Reporter`. Pro změnu, jakým způsobem je výsledek zpracováván, například okamžité zobrazování v GUI testovacího nástroje, se pouze nahradí třída `Reporter` třídou poděděnou.

Pro identifikaci grafického prvku, ke kterému se daný jev váže, slouží třída `Element`, která bude obsahovat základní důležité informace o daném prvku. Každé pravidlo může mít několik takových parametrů s různými názvy. Pro jejich získání se využije dynamičnosti jazyka a reflexí se vyberou všechny parametry správného typu. Stejný způsobem se budou vybírat parametry typu `RuleConfig`. Ty slouží ke konfiguraci vlastností některých pravidel.

Jedná se o třídní parametry a konkrétní algoritmus je může použít ještě předtím, než rozhodne, zda se jedná o porušení principů, či nikoliv. Příkladem takového nastavení může být počet položek v menu nebo maximální úroveň zanoření menu.

4.4 Report

Výsledný report z testování je třeba ukládat ve vhodném formátu a je žádoucí jej přehledným způsobem zobrazit uživateli. Zvolil jsem ukládaní ve formátu XML. Pro zobrazení se tento formát převede pomocí jazyka XSLT[1] do formátu, jenž bude vhodný pro zobrazení.

Tímto formátem může být například HTML. Případně by šlo zobrazit report i v GUI testovacího nástroje. Pro toto by bylo vhodné jej pomocí XSLT převést do formátu XML, který je používán pro tvorbu objektů GUI.

4.5 Návrh GUI

Jelikož nástroj slouží pro testování grafického uživatelského rozhraní, očekává se, aby měl také své vlastní, které by umožnilo jednodušší výběr testované aplikace než zadávání názvu programu do příkazového řádku. Je třeba zabránit neúmyslnému testování vlastního nástroje, ačkoliv by bylo zajímavé, kdyby šlo otestovat toto GUI samotným programem.

Program by měl více pracovat na pozadí, tudíž pro základní ovládání stačí tlačítko pro vybrání testované aplikace kliknutím do jejího okna. Dále by měl být zobrazen název testované aplikace. Poté již stačí jediné tlačítko pro spuštění samotného testování. Důležité je zobrazovat, co právě aplikace provádí a zda už uživatel může testovat svůj program. Pro tuto situaci se mi zdá vhodný stavový řádek.

[1] XSLT – Extensible Stylesheet Language Transformations slouží k převodu zdrojových dat ve formátu XML do libovolného výstupního formátu.

Kapitola 5

Implementace

Aplikace je založena na testovacím frameworku Dogtail (kapitola 2.3.3), který využívá knihovnu py-at-spi. Ta je také navíc využita pro sledování události ze systému. Jelikož je Dogtail implementován v jazyce Python, je využit pro vyvoření celého programu tento jazyk. Podle doporučení[1] je vhodné nové programy tvořit v Pythonu verze 3. Ačkoliv ve standardních repozitářích není dostupná portace Dogtailu pro Python 3, lze ji stáhnout z oficiální stránky projektu[2].

5.1 Získávání informací o elementech

Prvek uživatelského rozhraní v knihovně py-at-spi je reprezentován třídou `Accessible`, Dogtail tuto třídu dále rozšiřuje a označuje jako `Node`. Mezi nejdůležitější informace, které lze z této třídy získat, patří typ prvku označován jako role, dále jeho název, pozice, velikost i to, zda je prvek viditelný. Pokud je správným způsobem vyrobeno uživatelské rozhraní, lze zjistit i vztahy mezi jednotlivými prvky, například příslušnost popisku ke konkrétnímu prvku. Pro další speciální prvky je možno zjistit i vlastnosti, které nejsou přímo viditelné pro uživatele.

Přestože Dogtail poskytuje pro ovládání prvků i svůj procedurální přístup a nabízí podpůrné metody pro podporu testování, v mé práci z něj využiji pouze třídu související s třídou `Node`.

5.1.1 Identifikace

Pro identifikaci prvků jsem se rozhodl využít již dostupné prostředky z knihovny Dogtail, které umožňují sestavit absolutní cestu k danému elementu. Tato cesta je tvořena pomocí volání metod pro vyhledávání prvků, postupným zavoláním řetězce těchto metod se dá dostat až ke specifikovanému prvku. Pro zjednodušení ukládání jsem zvolil možnost vygenerování zdrojového kódu v Pythonu, který způsobí vyhledání prvku.

Protože není jisté, zda se cesta pro každé spuštění aplikace vygeneruje stejně, dokonce se může i drobně změnit uživatelské rozhraní, nepoužívá se pro ověření, jestli se jedná o stejný prvek porovnávání cesty, nýbrž se provede vyhledání a porovnávají se nalezené objekty. V případech, kdy se porušení pravidla týká dvou prvků, stačí pro ignorování celého jevu ignorovat pouze jediný prvek.

[1] Python Wiki: *Should I use Python 2 or Python 3 for my development activity?* [online] [cit. 2015-05-05]. Dostupné na: https://wiki.python.org/moin/Python2orPython3

[2] Dogtail: https://fedorahosted.org/dogtail/

Vyhledání probíhá postupným zavoláním metod s určitými parametry, tyto metody jsou zapsány jako řetězec, který je předán funkci `eval`, jež zadaný řetězec vykoná. Vkládaný kód není kontrolován, což může představovat bezpečnostní riziko.

5.2 Sledování událostí

Třída `EventListener` sleduje vybrané události[3] a dále je zpracovává. Pro události je při vytváření objektu zaregistrovaná obslužná rutina. Při zahájení testování je spuštěna metoda `pyatspi.Registry.start`. Jedná se o hlavní smyčku, která poté hlídá zaregistrované události. Třída `EventListener` ji spouští ve zvláštním vlákně, neboť z této smyčky se vystoupí až po zavolání `pyatspi.Registry.stop`. Jelikož opakované spouštění a ukončování této smyčky způsobovalo problémy s vícenásobným vyvoláním události, použil jsem pro zastavení sběru události boolovský parametr, jenž určuje, zda se události mají dále zpracovávat.

Statické kontroly jsem se rozhodl navázat na aktivaci okna testované aplikace, tím je myšleno přenesení okna do popředí, obvykle po kliku do okna aplikace nebo pomocí klávesové zkratky Alt + Tab. Třída `EventListener` si na základě události o aktivaci a deaktivaci okna ukládá informace o právě aktivním okně a pouze v případě změny vyvolá událost, která spustí kontrolu vybraných pravidel.

Základní událostí, kterou je třeba sledovat, je kliknutí na určitý prvek uživatelského rozhraní. Takto konkrétní událost není poskytována. Místo toho se odchytává událost kliknutí myši na pozici na obrazovce. Pomocí metody `getChildAtPoint` knihovny Dogtail se vyhledá potomek na pozici v aktuálním okně. Ve zmíněné metodě jsem objevil chybu, její popis a řešení naleznete v příloze B.

V případech, kdy je část okna překryta plovoucím objektem, například menu nebo výsuvný panel, je zmíněné řešení nedostatečné. Když uživatel klikne na objekt v daném panelu, tak se prvek vyhledá nesprávně v hlavním okně místo v plovoucím prvku. Řešení spočívá v udržování seznamu aktuálně zobrazených panelů na základě události o jejich zobrazení nebo skrytí. V tomto seznamu se poté přednostně vyhledává element, na který se kliklo. V některý případech se stane, že po kliknutí na prvek je tento prvek skryt. Typicky se skrývá celé menu po kliknutí na jednu položku nebo třeba dialogové okno po kliku na tlačítko. V takovém případě již nelze vyhledat prvek uživatelského rozhraní, jelikož neexistuje nebo je skryt, a informace o umístění již nejsou platné. Proto jsem přistoupil k řešení, kdy se zpracovává událost reagující na pohyb myši a pro každou novou pozici se vyhledá prvek. Z něj se poté zkopírují důležité informace jako je pozice, velikost a samotné uložení objektu v paměti.

Jelikož je prohledávání rekurzivní, mohlo by mít dopad na rychlost programu, z toho důvodu se toto prohledávání provádí pouze u plovoucích panelů. Položky menu se také ukládají před samotným kliknutím, v tomto případě je však využita událost, která nastane při označení položky, tím se ušetří vyhledávání prvku na pozici, tato událost totiž přímo předává objekt reprezentující položku menu.

[3] Tabulka všech dostupných událostí na adrese: http://accessibility.linuxfoundation.org/a11yspecs/atspi/adoc/atspi-events.html

5.3 Klasifikace pravidel

Protože závažnost v porušení pravidel je rozdílná, je důležité, aby pravidla byla rozdělena do skupin. Pokud je pravidlo striktně definováno a jeho porušení má vliv na pohodlné ovládání programu, je označeno za chybu. V případě, že je specifikováno vágněji nebo jeho porušení má zanedbatelný vliv na ovládání programu, značí se jako varování. Pravidla, která pouze doporučují, jsou označena za návrh. Každá kategorie má definovanou svou závažnost číselným ohodnocením, jenž se použije pro výpočet finálního skóre.

Rozdělení pravidel do těchto kategorií je provedeno pomocí dekorátorů[4]. Pro každou kategorii existuje třída, která když se použije k odekorování třídy s popisem pravidla, tak jej rozšíří o číselné ohodnocení a přidá informaci, jakého typu je dané pravidlo.

5.4 Snímky obrazovky

U některých chyb je pro názornější popis poskytován snímek obrazovky s vyznačenými elementy uživatelského rozhraní. Snímek obrazovky je získán při obsluze události a cesta k souboru je předána algoritmům, které kontrolují pravidla. Až po ukončení testování při generování reportu je v obrázku vyznačen prvek vykreslením rámečku okolo něj. Snímání obrazovky ve smyčce, která zpracovává události, by zdržovalo testování pravidel. Nejpomalejší částí je ukládání souboru na disk. Proto jsem upravil původní funkci z knihovny Dogtail tak, aby ukládání získaného snímku probíhalo až v samostatném vlákně.

5.5 Report

Report se vytváří ve třídě `Reporter`, která uchovává všechny zkontrolované jevy. Je docela možné, že některé kontroly proběhly opakovaně, proto je třeba ze seznamu vybrat množinu unikátních jevů. Jev je stejný, pokud má všechny prvky uživatelského rozhraní shodné, tato shoda je řešena porovnáním vyhledávací cesty.

Výsledný XML report obsahuje datum a čas, celkové skóre a popis chybných vlastností GUI. Nemá smysl uchovávat přesný popis správných situací, jelikož předpokládám, že jich bude mnohonásobně více. V elementu `description` je popis dané chyby. Každý jev však vychází z principů GUI, ty jsou popsány v kapitole Pravidla, a proto je vždy u každého pravidla uvedena citace z daných zdrojů [4, 11, 16]. Z důvodu, že tato citace patří k jednomu konkrétnímu typu pravidla, je uvedena ve výsledném reportu pouze jednou v sekci `reasons`. Součástí informací o porušení pravidla je i struktura (element `ignore`), kterou lze použít k ignorování tohoto konkrétního výskytu.

Kód 5.1: Struktura XML reportu

```
1 <report>
2   <date>2015-05-12 17:20:35</date>
3   <score>83.94160583941606</score>
4   <fails>
5     <rule id="139640270868608" name="MenuSize">
6       <flags>
7         <Suggest/>
8       </flags>
9       <description>
```

[4] Nejedná se o návrhový vzor dekorátor, nýbrž o zvláštní konstrukt z jazyka Python. Více v dokumentaci: https://docs.python.org/3.4/glossary.html#term-decorator.

```
10          Menu 'File' contains 1 items. Should be between 3 and 12
11        </description>
12        <ignore rule="MenuSize">
13          <elementpath name="menu">
14            path to element
15          </elementpath>
16        </ignore>
17      </rule>
18      ...
19    </fails>
20    <reasons>
21      <reason rule="MenuSize">
22        Menus should contain between three and 12 ...
23      </reason>
24      ...
25    </reasons>
26 </report>
```

5.5.1 Počítání skóre

Pro porovnání výsledků testování různých aplikací nebo více verzí stejné aplikace je užitečné reprezentovat jednotně celkový výsledek. Inspiroval jsem se nástrojem Pylint[5], který slouží pro statickou analýzu zdrojového kódu v jazyce Python. Tento program upozorňuje na chyby a porušení formátovacích pravidel. A výsledkem je skóre, kde maximální hodnota je 10 a minimální teoreticky neexistuje. Vzorec Pylintu (5.1) používá ohodnocení závažnosti pravidel.

$$10.0 - \left(\frac{5.0 \cdot error + warning + refactor + convention}{statement} \cdot 10 \right) \tag{5.1}$$

Jak již bylo zmíněno, pravidla jsou rozdělena do kategorií, které mají vlastní ohodnocení. U některých kontrolovaných jevů je velmi malá pravděpodobnost výskytu chyby na počet kontrol, tudíž odhalení chyby skóre by změnilo skóre jen minimálně. Tento typ chyby má nastaveno skóre asymetricky, kdy za dodržení pravidla je přiděleno méně bodů, než je ztraceno za jeho porušení. Tím docílíme toho, že i jediný výskyt takového pravidla způsobí viditelnou změnu skóre. Implementačně to funguje tak, že u každého pravidla je podle jeho zařazení třídy závažnosti nastaveno skóre jak pro úspěch, tak selhání. Pokud je třeba na některé pravidlo aplikovat asymetrické rozložení, lze jej navíc odekorovat třídou `Rare`. Výsledné skóre se počítá podle vzorce 5.2 a maximální hodnota je 100.

$$100 - \left(\frac{\sum\limits_{chybné\ jevy} body\ za\ chybu}{\sum\limits_{správné\ jevy} body\ za\ úspěch + \sum\limits_{chybné\ jevy} body\ za\ úspěch} \right) \cdot 100 \tag{5.2}$$

[5] http://www.pylint.org/

Kapitola 6

Testování aplikací za použití implementovaného nástroje

Pouze aplikace, které správně podporují asistivní technologie, jsou implementovaným nástrojem testovatelné. Pro ověření funkčnosti testovaných vlastností je potřeba nalézt programy, které tyto chyby obsahují a jsou testovatelné. Pro některé jevy bylo třeba vyrobit ukázkové programy.

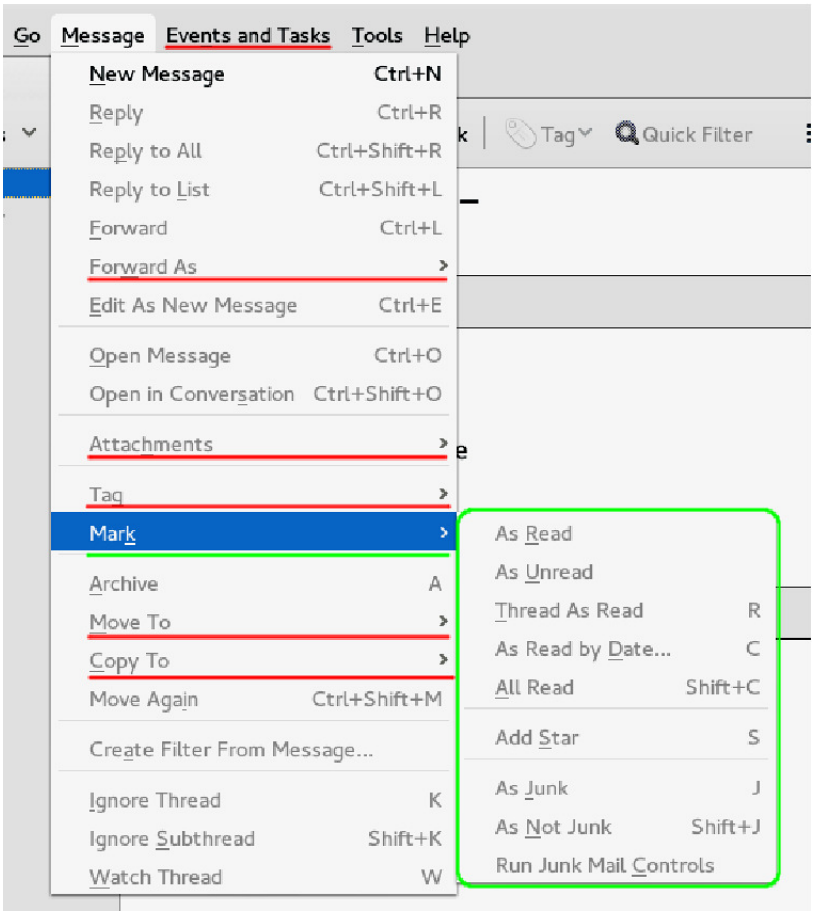
Experimentování probíhalo v operačním systému Fedora 21 s prostředím GNOME. Aplikace je funkční i v systému Ubuntu s prostředím Unity, kde jsou ovšem některá pravidla netestovatelné vlivem vlastností tohoto prostředí. Mezi významnou změnu v Unity patří přesun hlavního menu z aplikace do systémové lišty. Z hlediska struktury uživatelského rozhraní není v tomto případě vlastníkem menu aplikace, nýbrž systém. Aplikace je také využitelná v prostředí KDE, zde je ovšem potřeba nastavit systém podle návodu z knihovny Dogtail. I přes různé nastavení se nepodařilo otestovat programy vytvořené pomocí toolkitu Qt, neboť výsledný program vůbec nespolupracoval s knihovnou AT-SPI.

Při spuštěném testování se ve stavové liště píše počet unikátních otestovaných jevů (obrázek 6.2). Vhodné použití je ovládat testovanou aplikaci tak dlouho, dokud se počet otestovaných jevů nemění.

6.1 Thunderbird

Emailový klient Thunderbird je příkladem komplexní aplikace s rozsáhlým uživatelským rozhraním. Mnoho falešně negativních nálezů vzniklo z důvodu používání tlačítka s jiným vzhledem. Celkový výsledek prvního testování byl 84,8. Po přidání pravidel pro velikost tlačítek do seznamu k ignorování byl výsledek 88,1.

Počet zkontrolovaných jevů se pohybuje okolo 1300. Mezi časté chyby patří deaktivování menu, když obsahuje další položky. Uživatel tudíž nemá možnost zjistit, jaké volby jsou mu skryty. Na obrázku 6.1 je vidět tento jev celkem pětkrát špatně, zeleně je vyznačeno správné řešení. I když jsou všechny položky v podmenu neaktivní, pořád je lze zobrazit. Závažnou chybou je také označení „Event and Task“ v hlavním menu, kde by měly být pouze jednoslovné názvy. Za zmínku stojí, že v dřívější verzi bylo použito označení „Calendar“. Mezi časté chyby také patří přiřazení stejné *Access Key* více prvkům.



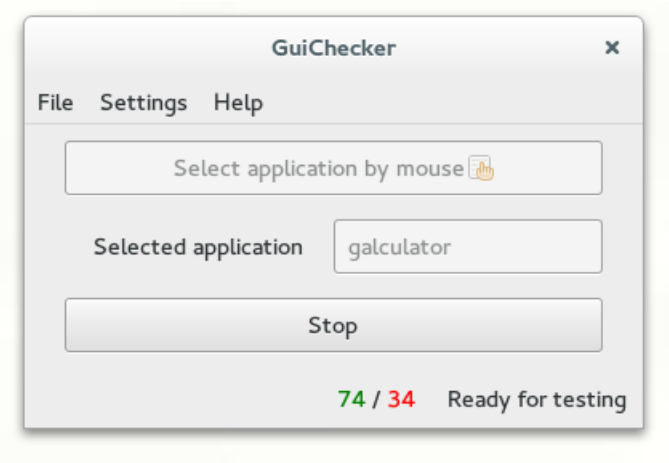
Obrázek 6.1: Ukázka chyb v menu Thunderbirdu. Barevné čáry nejsou výsledkem testovacího programu.

6.2 Galculator

Standardní kalkulačka v prostředí GNOME je naopak ukázkou velmi jednoduché aplikace. Ačkoliv jsou nalezené chyby málo závažné, je dosažené skóre (76,3) nízké. Je to důsledek zkontrolování pouze 100 prvků, z nichž u 35 byla nalezena drobná připomínka.

6.3 Vlastní aplikace

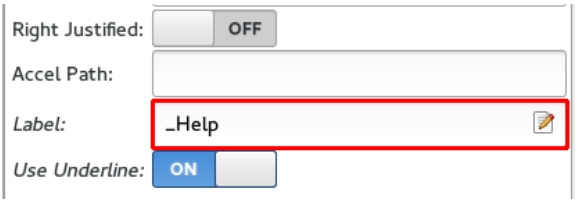
Aplikace pro testování má možnost sama sebe testovat. Protože jsem měl možnost uživatelské rozhraní měnit, tak obsahuje pouze jediný drobný nedostatek a to málo položek v menu „Help“. Při průběžném testování byly odhalené chyby jako chybějící nebo neunikátní *Access Key*, špatné použití výpustky nebo chybějící popisek.



Obrázek 6.2: Výsledná aplikace v průběhu testování.

6.4 Glade

Jak již bylo zmíněno, Glade je nástroj pro tvorbu uživatelského rozhraní. Kromě špatně označených chyb je jediná závažná chyba v použití *Access Key* pro více prvků. Na obrázku 6.3 je označeno textové pole, kterému chybí popisek. Na první pohled se jedná o špatné označení, ovšem za zmínku stojí, že ostatní pole jako chybně označená nejsou. To je způsobeno tím, že tyto informace se berou z pohledu asistivních technologií. Při tvorbě tohoto rozhraní bylo zapomenuto přiřadit správně popisek pouze u tohoto jednoho pole.



Obrázek 6.3: Výřez ze snímku obrazovky pořízeném testovacím nástrojem.

6.5 Gedit

Gedit je standardní textový editor v GNOME, v nové verzi už neobsahuje ani hlavní menu a jedná se o velmi jednoduchou aplikaci. Zkontrolováno bylo přibližně 140 elementů a nebyla objevena žádná závažná chyba.

Kapitola 7

Závěr

Cílem práce bylo vytvořit nástroj pro kontrolu správných vlastností uživatelského rozhraní. Analyzováním vhodnosti a náročnosti automatizace u principů uživatelského rozhraní pro různá prostředí byl vytvořen seznam vlastností a pravidel, z nichž jsem některé implementoval, nicméně pro větší užitečnost programu by bylo vhodné kontrolovat mnohem více pravidel, jejichž přidávání umožňuje navržená objektová architektura.

Výsledný nástroj kontroluje uživatelské rozhraní na základě ovládání testované aplikace uživatelem (testerem). Pro zautomatizování tohoto testování by bylo potřeba podstatně změnit návrh aplikace, který by ovšem také vycházel ze sestaveného seznamu pravidel. Testování by bylo vhodné rozšířit o analyzování pokrytí otestovaných vlastností uživatelského rozhraní podle různých metrik. Počet otestovaných vlastností je v tomto ohledu nedostatečný.

Program nemůže odhalit logické nedostatky a špatně navrženou strukturu ovládání programu, pouze upozorňuje na možné chyby a pomáhá odhalit drobnosti, které návrhář přehlédl. Aplikaci nelze použít pro porovnání kvality uživatelského rozhraní na základě výsledného skóre, to slouží spíše pro porovnávání jednotlivých verzí stejného programu. Hodně odhalených chyb je označeno špatně, můžeme je však nechat ignorovat pro další testování. Výsledek programu se dá využít k opravě drobných nedostatků a úpravě některých řešení. Při vývoji uživatelského rozhraní je vhodné jej průběžně kontrolovat tímto nástrojem. Tento způsob byl také využit při vývoji a návrhu rozhraní samotného nástroje, kdy pro testování byla použita prozatímní verze, která pracovala z příkazového řádku.

Část aplikace, jež zpracovává události, by se dala využít pro vytvoření programu pro podporu automatizace s knihovnou Dogtail. Taková aplikace by sloužila k nahrávání akcí na prvky uživatelského rozhraní a vytvářela by kód pro spuštění nahrané sekvence. Na stejném principu funguje Selenium IDE, které slouží pro automatizování webových stránek.

Literatura

[1] GUI Definition. *The Linux Information Project*, [online] [cit. 2015-05-10].
Dostupné na: http://www.linfo.org/gui.html

[2] Selenium – Web Browser Automation. [online] [cit. 2015-05-10].
Dostupné na: http://www.seleniumhq.org/

[3] Ackermann, P.; Velasco, C. A.; Power, C.: Developing a Semantic User and Device Modeling Framework That Supports UI Adaptability of Web 2.0 Applications for People with Special Needs. In *Proceedings of the International Cross-Disciplinary Conference on Web Accessibility*, W4A '12, New York, NY, USA: ACM, 2012, ISBN 978-1-4503-1019-2, s. 12:1–12:4, doi:10.1145/2207016.2207018.
Dostupné na: http://doi.acm.org/10.1145/2207016.2207018

[4] Apple Inc.: OS X Human Interface Guidelines. [online] [cit. 2015-01-22].
Dostupné na: https://developer.apple.com/library/mac/documentation/UserExperience/Conceptual/OSXHIGuidelines/index.html

[5] Chang, T.-H.; Yeh, T.; Miller, R. C.: GUI Testing Using Computer Vision. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, New York, NY, USA: ACM, 2010, ISBN 978-1-60558-929-9, s. 1535–1544, doi:10.1145/1753326.1753555.
Dostupné na: http://doi.acm.org/10.1145/1753326.1753555

[6] Feldman, E.: Create User Interfaces with Glade. *Linux J.*, ročník 2001, č. 87, Červenec 2001: s. 4–, ISSN 1075-3583.
Dostupné na: http://dl.acm.org/citation.cfm?id=509446.509450

[7] Hartson, H.; Pyla, P. S.: *The UX Book*. Boston: Elsevier, 2012, ISBN 01-238-5241-2.

[8] Hocke, R.: Sikuli Documentation for version 1.1+. [online] [cit. 2015-05-10].
Dostupné na: http://sikulix-2014.readthedocs.org/en/latest/index.html

[9] Jang, Y.; Song, C.; Chung, S. P.; aj.: A11Y Attacks: Exploiting Accessibility in Operating Systems. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, New York, NY, USA: ACM, 2014, ISBN 978-1-4503-2957-6, s. 103–115, doi:10.1145/2660267.2660295.
Dostupné na: http://doi.acm.org/10.1145/2660267.2660295

[10] jQWidgets: Advanced jQuery and HTML5 UI framework. [online] [cit. 2015-05-05].
Dostupné na: http://www.jqwidgets.com/

[11] KDE TechBase: KDE Human Interface Guidelines. [online] [cit. 2015-01-22]. Dostupné na: https://techbase.kde.org/Projects/Usability/HIG

[12] Microsoft: Design apps for the Windows desktop. [online] [cit. 2015-04-25]. Dostupné na: https://msdn.microsoft.com/en-us/windows/desktop/aa511258.aspx

[13] Shah, N.; Balda, G.: *HTML5 Enterprise Application Development*. Birmingham: Packt Publishing, 2013, ISBN 9781849685689.

[14] Telerik: Kendo UI. [online] [cit. 2015-05-05]. Dostupné na: http://www.telerik.com/kendo-ui

[15] The Glade project: Glade - A User Interface Designer. [online] [cit. 2015-05-07]. Dostupné na: http://glade.gnome.org/

[16] The GNOME Project: GNOME Human Interface Guidelines. [online] [cit. 2015-01-22]. Dostupné na: https://developer.gnome.org/hig/3.14/

[17] The GTK+ Team: The GTK+ Toolkit. [online] [cit. 2015-05-07]. Dostupné na: http://www.gtk.org/

[18] Yosifovich, P.: *Windows Presentation Foundation 4.5 Cookbook*. Packt Publishing, 2012, ISBN 184968622X.

Příloha A

Obsah DVD

- **tex** – zdrojové soubory s textem této práce
- **bp.pdf** – text této práce ve formátu pdf
- **guichecker** – zdrojové soubory programu
  Dostupné také z: https://bitbucket.org/vkreuz/guichecker
- **fedora.ova** – virtuální stroj se systémem Fedora a připraveným prostředím pro běh aplikace
- **dogtail** – knihovna Dogtail, na které je program závislý
  Dostupné z: https://fedorahosted.org/dogtail/

Příloha B

Chyby v Dogtailu

V průběhu vývoje jsem objevil dvě chyby v knihovně Dogtail. Obě jsem nahlásil vývojářům. Opravy chyb jsou součástí souboru `dogtailfix.py` a jsou dynamicky nahrány místo původních metod.

makeScriptMethodCall https://bugzilla.redhat.com/show_bug.cgi?id=1207841

Metoda `GenericPredicate.makeScriptMethodCall` má za úkol vytvořit kód v jazyce Python, který provede vyhledání prvku uživatelského rozhraní. Chyba spočívá ve vynechání oddělovací čárky v parametru funkce.

Aktuální výsledek: `child( name="Name"roleName='Role', recursive=False)`

Očekávaný výsledek: `child(name="Name", roleName='Role', recursive=False)`

getChildAtPoint https://bugzilla.redhat.com/show_bug.cgi?id=1215679

Při volání metody `getChildAtPoint` se program dostal do nekonečné smyčky. Tato událost nastává pouze při vyhledávání na určité pozici v programu Thunderbird.

Oprava spočívá v přidání podmínky na řádku 7, protože v tomto případě místo vrácení hodnoty `None` pro označení, že nic nebylo na dané pozici nalezeno, byl metodou `getAccessibleAtPoint` vrácen stejný uzel.

Kód B.1: Oprava chyby

```
1  def _get_child_at_point(self, x, y):
2      node = self
3      while True:
4          try:
5              child = (node.queryComponent()
6                       .getAccessibleAtPoint(x, y, pyatspi.DESKTOP_COORDS))
7              if child == node:
8                  break
9              elif child and child.contains(x, y):
10                 node = child
11             else:
12                 break
13         except NotImplementedError:
14             break
15     if node and node.contains(x, y):
16         return node
17     else:
18         return None
```

Příloha C

Implementovaná pravidla

Seznam implementovaných pravidel tak, jak jsou popsána v programu.

Ellipses Use an ellipsis (...) at the end of a label if the action requires further input from the user before it can be carried out. Do not add an ellipsis to commands like Properties or Preferences.

MenuGroupSize Organize the menu items into groups of seven or fewer strongly related items.

MenuNesting Avoid using more than one level of submenus. A submenu that contains other submenus can be difficult for users to use. Submenus should contain between three and six items, and should never contain other submenus.

MenuBarMultipleWordsTitle Menu titles on a menubar are single words with their first letter capitalized. Do not use spaces in menu titles, as this makes them easily-mistaken for two separate menu titles. Do not use compound words (such as WindowOptions) or hyphens (such as Window-Options) to circumvent this guideline.

TabsNesting Do not nest tabs.

MenuTitleDisable Do not disable menu titles. Allow the user to explore the menu, even though there might be no available items on it at that time.

AccessKeyNotUnique Access key must be unique

ButtonSize Do not use more than one or two different widths of button in the same window, and make all of them the same height. This will help give a pleasing uniform visual appearance to your window that makes it easier to use.

AccessKeyMissing Where possible, all labelled componentsshould have an access key.

LabelMising Label every line edit with a descriptive caption to the left. Label the combo box with a descriptive caption to the left. etc.

MenuSize Menus should contain between three and 12 top-level items. If a menu contains more than 12 items, evaluate whether all the items are necessary and belong in the menu. If you are unable to reduce the size, submenus can be used. However, they should be avoided if at all possible, as they are physically difficult to use. Submenus should contain between three and six items, and should never contain other submenus.

MenuBarMoreMenus Do not have more than nine menu categories within a menu bar. Too many categories are overwhelming and make the menu bar difficult to use.

ButtonDifferentSizeInRow When several buttons are placed next to each other, ensure that they have the same width. This is particularly important for pairs of Cancel and OK buttons.

HeaderCapitalization Header capitalization should be used for any headings, including header bar headings and page, tab and menu titles. It should also be used for short control labels that do not normally form proper sentences, such as button labels, switch labels and menu items.

MenuGroupMixedItems Do not mix different types of menu item within each group - actions, check box and radio button items should be kept separate.